

**Space Physics Interactive Data Resource
SPIDR**

**Web Services Guide
REST API v1,v2**

**Version 3.0
May 2010**

Contents

Contents	2
Introduction	3
Data Sources	3
Data Source Interface	3
Basic REST API Summary.....	4
Obtaining Images.....	4
Obtaining Data.....	7
Available Data Parameters to GetData	9
Obtaining Metadata	11
Available Metadata Parameters to GetMetadata	12
Example Basic REST Clients	13
IDL Rest Client.....	14
MATLAB REST Client.....	14
Python REST Client	14
Perl REST Client	14
Advanced Usage	15
Common Data Model	15
Common Query Language.....	16
Asynchronous RESTful Data Service	17
SPIDR Advanced RESTful Web Service By Example	20
Metadata	20
Synchronous get data.....	20
Asynchronous Data Requests	22

Introduction

This document is intended for software developers who are creating client applications that connect to SPIDR, and for end users who wish to know more about these interfaces into SPIDR (people using one of the existing Web Service clients for example). This document gives an architectural overview of SPIDR's RESTful web service implementation, to give a sense of what you as a client developer or user are interacting with, as well as detailed information for each service currently provided. SPIDR provides a RESTful API that includes a variety of capabilities described below, designed as a means for easy programmatic access to its data and metadata holdings. If you need to create a simple script or a more complicated client to obtain data or metadata from SPIDR, this document provides details and examples to do just that. There are several existing clients written in various programming languages that you may use as a starting point as well, referenced below. If you are a SPIDR user interested in general help or the web interface, please refer to the normal SPIDR User's Guide, or SPIDR's help features.

Data Sources

One of the main features of this interface is the wide range of supported data sources, including:

- Weather reanalysis databases NCEP25 and ERA40.
- Weather forecast from NWS.
- DMSP satellite data granule databases from NOAA.
- Several solar image databases from NOAA.
- Sea Surface Temperature (SST) gridded database from NOAA.
- Generic frontend to SPIDR databases from NOAA.

Data Source Interface

All database connection classes are programmed in Java and implement the common Data Source Interface (DSI), which has the following methods:

- `boolean isSupportCdm()` – shows if the resource returns CDM-compatible data. If so, further transformation of data is possible.
- `org.w3c.dom.Document getMetadata()` – returns the data resource metadata.
- `DataRequestBean getData(DataRequestBean request)` – the most common method for data retrieving from data source.
- `DataRequestBean downloadData(DataRequestBean request)` – the method is used for transferring large amounts of data, mostly asynchronously. Stores data to a storage in the requested format and returns the link or another ID, which helps user to download data.
- `DataRequestBean shiftData(DataRequestBean request)` – a method

to work with data that can be represented in granule format (e.g. orbital segments). The call moves the current viewable granule object to another one relatively to current.

- `DataRequestBean viewData(DataRequestBean request)` – requests the visualization of data. Each data resource has its own format of visualization.
- `String outputData(Object data, String format)` – converts the data to a requested format. The method is usually called inside an activity when a specific data output format is requested.

This limited number of functions simplifies the data queries to multiple resources using either SOAP requests to the OGSA-DAI activities or REST web-service interface to the OGSA-DAI engine.

Basic REST API Summary

The REST interface provides the means to obtain data and metadata, but it also has the ability to provide high quality image plots (PNG format). REST calls are very straight forward conceptually, and amount to constructing a URL with the appropriate fields. Creating a REST call for SPIDR consists of formulating your base URL, and then adding the additional pieces you need to accomplish your desired goal.

The base URL will typically be `http://spidr.ngdc.noaa.gov/spidr/servlet/`, or will reference one of SPIDR's other worldwide mirrors. What you tack on to the base URL will vary based on whether you want data, metadata, or images, and for which parameters and when. Each URL is composed of three pieces common to both data and metadata, however parameters will vary. URL structure is therefore comprised of a prefix and method, followed by a query string which is a list of parameters that make up the call. A few are required, and some are optional. At the highest level, you will structure your URLs as follows:

`<prefix>/<method>?<query_string>`

The values you use for each of these pieces are detailed below, based on the type of call you're making.

Obtaining Images

Recall the template

`<prefix>/<method>?<query_string>`

Which in the case of image plots relies on the `GetData` method, so the first two components of the call would be as follows (assuming you're using the primary site)

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?>

All that remains is the query string, and the following table details the required and optional fields for this call (bold italic are required).

Table 20 – GetData, image parameters

Parameter	Description
Width	Width of plot, in pixels. Optional. If not specified, the plot is auto-scaled.
Height	Height of plot, in pixels. Optional. If not specified, the plot is auto-scaled
marks	Optional. Allows one to specify non-default attributes of a plot. Allowed values: dots, points, none
representation	Optional. Allowed values: bars, line (default)
color	Optional. Only applies to format=image If not specified, SPIDR will choose. A semi-colon delimited list (1-1 for param items) of hexadecimal color definitions. e.g. param=foF2.BC840 => color=0x00ff00 param=foF2.BC840;foF2.WP937 => color=0xff00ff;0x00ff00
<i>format</i>	<i>Required. For image plotting, the only available value is 'image'</i>
<i>dateFrom</i>	<i>Required. Defines the start time from which to obtain data</i>
<i>dateTo</i>	<i>Required. Defines the stop time to obtain data until.</i>
<i>param</i>	<i>Required. Defines the data set from which data are obtained</i>

One nice thing about RESTful web services is that examples can easily be provided through a web browser. Here are several examples that illustrate some of the capabilities of the REST web services. Using the table above, you can construct an image with varying height, width, coloring, and combined data sets.

Image/Plotting Examples

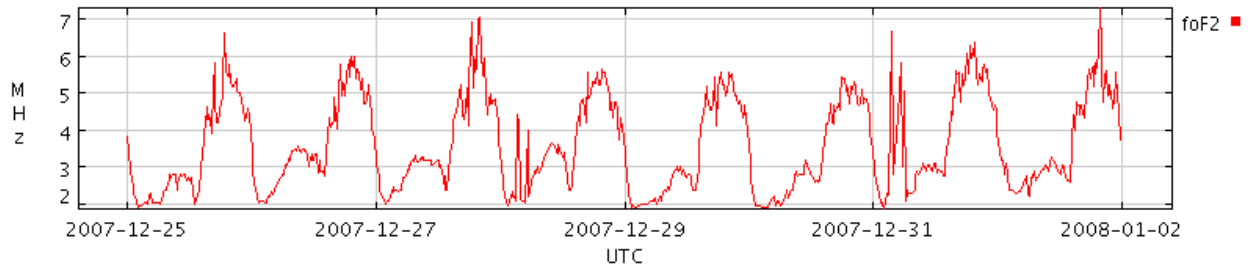
A line plot of the foF2 ionospheric parameter for the Boulder ionosonde, with the following call parameters for the <query_string> portion of the template

- format=image
- param=foF2.BC840
- dateFrom=20071225
- dateTo=20080101
- marks=none
- height=200

Complete URL/call

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?format=image¶m=foF2.BC840&dateFrom=20071225&dateTo=20080101&marks=none&height=200>

Which produces the following image

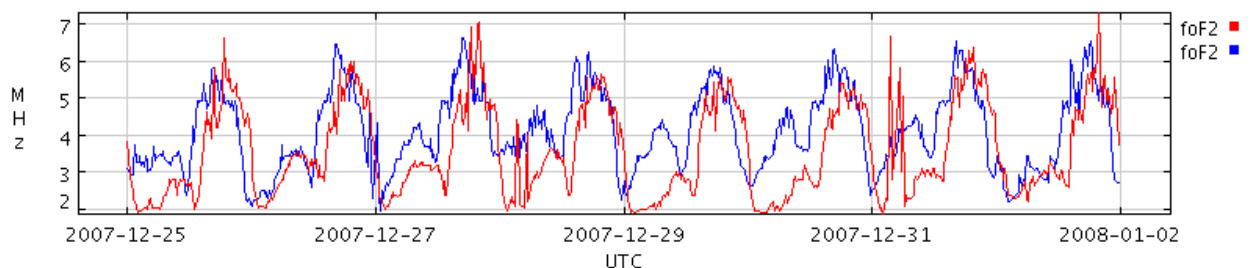


The same parameter, for the same period of time, only for two stations this time, Boulder and Wallops Island, with the following call parameters for the <query_string> portion of the template

- format=image
- param=foF2.BC840;foF2.WP937
- dateFrom=20071225
- dateTo=20080101
- marks=none
- height=200

Complete URL/call

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?format=image¶m=foF2.BC840;foF2.WP937&dateFrom=20071225&dateTo=20080101&marks=none&height=200>

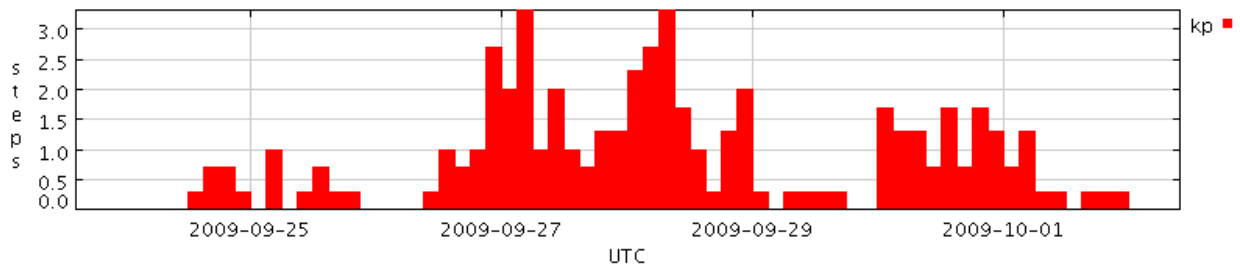


Here's a bar chart of KP, using the following parameters in the <query_string> portion of the template

- format=image
- param=index_kp
- dateFrom=20090924
- dateTo=20091001
- representation=bars
- height=200

Complete URL/call

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?format=image¶m=index_kp&dateFrom=20090924&dateTo=20091001&representation=bars&height=200

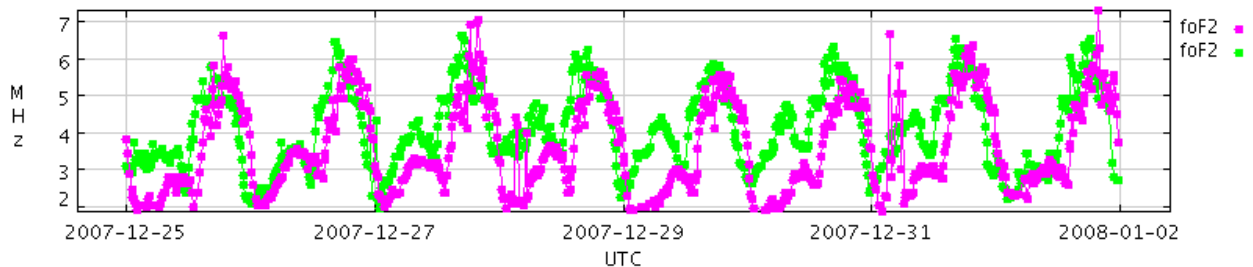


And finally, a line plot with marks, for the same data as the other ionosonde examples, with the following parameters (to help visualize outliers for example), and custom colors filling out the <query_string> portion of the template

- format=image
- param=foF2.BC840;foF2.WP937
- dateFrom=20071225
- dateTo=20080101
- marks=dots
- height=200
- color=0xff00ff;0x00ff00

Complete URL/call

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?format=image¶m=foF2.BC840;foF2.WP937&dateFrom=20071225&dateTo=20080101&marks=dots&height=200&color=0xff00ff;0x00ff00>



Obtaining Data

Again, recall the REST call template

<prefix>/<method>?<query_string>

In the case of data, it relies on the GetData method, so the first two components of the call would be as follows (assuming you're using the primary site)

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?>

All that remains to complete the call is the query string, and the following table details the required and optional fields for this call (bold italic are required).

Table 21 – GetData data parameters

Parameter	Description
<i>format</i>	<p>Required. For obtaining data, there are several format options available currently, and they change the behavior of the response as well as the data given.</p> <p>The available values are:</p> <p>‘matlab’ – returns a URL to a zip file containing data files formatted for use by matlab applications</p> <p>‘xml’ –returns a URL to a zip file containing data in XML format</p> <p>‘zip’ – returns a URL to a ZIP file containing files in SPIDR’s proprietary ASCII format</p> <p>‘ascii’ – returns a data stream containing the requested data in SPIDR’s proprietary ASCII format.</p> <p>‘csv’ – returns a data stream containing the requested data in CSV format, with the following fields:</p> <p>time, value, qualifier, description</p> <p>not all fields are applicable to all data sets though.</p>
<i>dateFrom</i>	Required. Defines the start time from which to obtain data
<i>dateTo</i>	Required. Defines the stop time to obtain data until.
<i>param</i>	Required. Defines the data set from which data are obtained. Full enumeration in the next section.

The GetData call has a field called ‘param’ which determines the dataset from which data are obtained. SPIDR has dozens of datasets, many of which have multiple formats, so it can be difficult to determine which dataset you’re after. This is where the self describing feature of the REST web services come into play. You can query the GetData service to find out which parameters it supports. The param field is a period delimited triplet composed of ‘param.platform.section’. Not all params have all three components, some are just the first. Other datasets provide optional drill down capabilities to select a specific station and data format via

this triplet. The best way to find out more about the param is to query the GetData service by passing it a 'describe' option. More details regarding this follow.

Available Data Parameters to GetData

At present, all of SPIDR's time series data are available from the RESTful web service interface, which amounts to 220 different possible parameters (not including narrowing elements). As stated, the parameter field consists of a triplet of 'param.platform.section'. In the cases where only one component of the triplet is defined, e.g. index_kp, no other pieces are needed or used, 'param=index_kp' is all that's allowed. In other words, the other components of the triplet allow you to narrow your results for datasets where narrowing is supported. The full mapping for 'param=' is too large to include here, so it is available dynamically from the GetData service itself, via a call to: <http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?describe> You can further query, with a parameter and portions of the triplet where available, to find more information including which stations are available, which sections, and which time ranges.

The describe call allows you to drill down by providing a parameter for which you want more information. For example, if you know you want the foF2 ionosphere parameter, but you don't know which stations are available, you can make a describe call to obtain the list of available stations. If you know that you want foF2 and the BC840 station for example, you can call describe with param=foF2.BC840 to find the available time range for that data set.

The GetData describe call supports three tiers of information. The first tier returns all of the available parameters. e.g.

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?describe>

The second tier returns available stations, based on a given parameter. e.g.

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?describe¶m=crl>

The third tier supplies the available time range for a given parameter + station. e.g.

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?describe¶m=crl.BKSN>

Using all of this information, constructing a complete call to obtain data is fairly straightforward, the hardest part is figuring out how to access the dataset you're after, which as described above, is where the describe feature is used. Below are some specific examples for obtaining data.

Data Examples

All of the data examples will focus on the 'ascii' and 'csv' formats. The default zip file format doesn't have much to show, since it merely responds with a URL from which you download the file that contains the same data as the 'ascii' call.

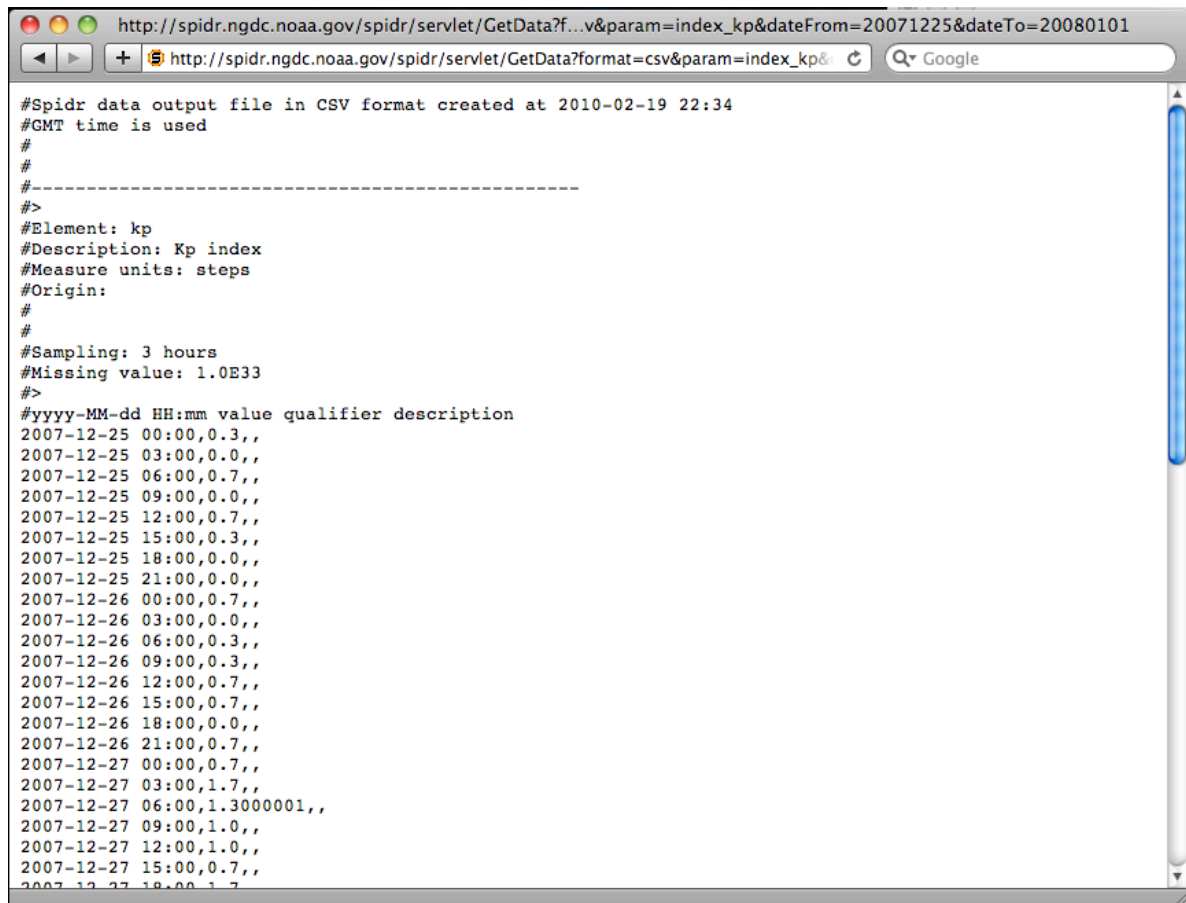
To begin, here's a small chunk of Kp index data, in CSV format, with the components of the <query_string> portion of the template:

- format=csv
- param=index_kp
- dateFrom=20090924
- dateTo=20091001

Complete URL/call

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?format=csv¶m=index_kp&dateFrom=20071225&dateTo=20080101

Which will give you something similar to the following if you open the URL in your web browser.

A screenshot of a web browser window showing the output of a SPIDR data query. The address bar displays the URL: http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?format=csv¶m=index_kp&dateFrom=20071225&dateTo=20080101. The page content is a text-based CSV output. It starts with a header line: "#Spidr data output file in CSV format created at 2010-02-19 22:34". This is followed by several lines of metadata: "#GMT time is used", "#", "#>", "#Element: kp", "#Description: Kp index", "#Measure units: steps", "#Origin:", "#", "#", "#Sampling: 3 hours", "#Missing value: 1.0E33", "#>". The main data is presented in a table with columns: "#yyyy-MM-dd HH:mm value qualifier description". The data rows show Kp index values for various dates and times, such as "2007-12-25 00:00,0.3,,", "2007-12-25 03:00,0.0,,", "2007-12-25 06:00,0.7,,", "2007-12-25 09:00,0.0,,", "2007-12-25 12:00,0.7,,", "2007-12-25 15:00,0.3,,", "2007-12-25 18:00,0.0,,", "2007-12-25 21:00,0.0,,", "2007-12-26 00:00,0.7,,", "2007-12-26 03:00,0.0,,", "2007-12-26 06:00,0.3,,", "2007-12-26 09:00,0.3,,", "2007-12-26 12:00,0.7,,", "2007-12-26 15:00,0.7,,", "2007-12-26 18:00,0.0,,", "2007-12-26 21:00,0.7,,", "2007-12-27 00:00,0.7,,", "2007-12-27 03:00,1.7,,", "2007-12-27 06:00,1.3000001,,", "2007-12-27 09:00,1.0,,", "2007-12-27 12:00,1.0,,", "2007-12-27 15:00,0.7,,", "2007-12-27 18:00,1.7,,".

```
#Spidr data output file in CSV format created at 2010-02-19 22:34
#GMT time is used
#
#-----
#>
#Element: kp
#Description: Kp index
#Measure units: steps
#Origin:
#
#
#Sampling: 3 hours
#Missing value: 1.0E33
#>
#yyyy-MM-dd HH:mm value qualifier description
2007-12-25 00:00,0.3,,
2007-12-25 03:00,0.0,,
2007-12-25 06:00,0.7,,
2007-12-25 09:00,0.0,,
2007-12-25 12:00,0.7,,
2007-12-25 15:00,0.3,,
2007-12-25 18:00,0.0,,
2007-12-25 21:00,0.0,,
2007-12-26 00:00,0.7,,
2007-12-26 03:00,0.0,,
2007-12-26 06:00,0.3,,
2007-12-26 09:00,0.3,,
2007-12-26 12:00,0.7,,
2007-12-26 15:00,0.7,,
2007-12-26 18:00,0.0,,
2007-12-26 21:00,0.7,,
2007-12-27 00:00,0.7,,
2007-12-27 03:00,1.7,,
2007-12-27 06:00,1.3000001,,
2007-12-27 09:00,1.0,,
2007-12-27 12:00,1.0,,
2007-12-27 15:00,0.7,,
2007-12-27 18:00,1.7,,
```

The next example is the same data set, in SPIDR's legacy proprietary ASCII format with the following <query_string> components.

- format=ascii
- param=index_kp
- dateFrom=20090924
- dateTo=20091001

Complete URL/call

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData?format=ascii¶m=index_kp&dateFrom=20071225&dateTo=20080101

Which will give you something similar to the following if you open the URL in your web browser.

```

#Spidr data output file in ASCII format created at 2010-02-19 23:02
#GMT time is used
#
#-----
#>
#Element: kp
#Description: Kp index
#Measure units: steps
#Origin:
#
#
#Sampling: 3 hours
#Missing value: 1.0E33
#>
#yyyy-MM-dd HH:mm value qualifier description
2007-12-25 00:00 0.3 "" ""
2007-12-25 03:00 0.0 "" ""
2007-12-25 06:00 0.7 "" ""
2007-12-25 09:00 0.0 "" ""
2007-12-25 12:00 0.7 "" ""
2007-12-25 15:00 0.3 "" ""
2007-12-25 18:00 0.0 "" ""
2007-12-25 21:00 0.0 "" ""
2007-12-26 00:00 0.7 "" ""
2007-12-26 03:00 0.0 "" ""
2007-12-26 06:00 0.3 "" ""
2007-12-26 09:00 0.3 "" ""
2007-12-26 12:00 0.7 "" ""
2007-12-26 15:00 0.7 "" ""
2007-12-26 18:00 0.0 "" ""
2007-12-26 21:00 0.7 "" ""
2007-12-27 00:00 0.7 "" ""
2007-12-27 03:00 1.7 "" ""
2007-12-27 06:00 1.3000001 "" ""
2007-12-27 09:00 1.0 "" ""
2007-12-27 12:00 1.0 "" ""
2007-12-27 15:00 0.7 "" ""
2007-12-27 18:00 1.7 "" ""

```

If you're writing a client application that will be consuming these data, the paradigm for reading the data is the same. Any programming language or library that supports an HTTP GET is able to make these calls and obtain and use the resulting data.

Obtaining Metadata

Again, recall the REST call template

<prefix>/<method>?<query_string>

In the case of Metadata, it relies on the GetMetadata method, so the first two components of the call would be as follows (assuming you're using the primary site)

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetMetadata?>

The metadata self describe has less information than data, since the GetMetadata call links to metadata records instead of actual data. There is only one format, XML, currently adhering to the FGDC schema standard. A param field is required to these calls.

Table 22 – GetMetadata parameters

Parameter	Description
-----------	-------------

<i>param</i>	<i>Required. Defines the document from which metadata are obtained. The detailed mapping of what is supported is below.</i>
--------------	---

Available Metadata Parameters to GetMetadata

Similarly to GetData, GetMetadata allows you to query for additional information to get the metadata record you're after. The topmost starting point is similar to GetData, simply supply the call with 'describe' without any additional elements in the query string and you'll get the topmost information back.

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetMetadata?describe>

Just like with the GetData call, you can provide a parameter to the describe call to drill down and obtain more specific information about specific stations. e.g.

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetMetadata?describe¶m=cri>

Unlike data, the GetMetadata call doesn't have a third tier of metadata for a platform or section, as it doesn't pertain to metadata. Only data have an associated time range, which is what the third tier supports. The GetMetadata call will accept such requests, but suggests that you probably wanted the associated GetData call instead. e.g.

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetMetadata?describe¶m=cri.BKSN>

In some cases there are differences between the theme/param in GetData and GetMetadata. For example, 'foF2' (an ionospheric parameter) vs 'iono'. The reason the metadata param value isn't 'foF2.BC840' is that foF2 isn't specific to the Boulder station. These parameters are designed to allow you to obtain unique information about the data sets, and we are exploring what way might be best to support the other param values, while still supplying useful information. These allow you to request metadata about the particular resources you're currently obtaining data for. For example, if you're requesting 'foF2.BC840' in a data call, you'll want 'iono.BC840' metadata. The responses are all in XML format, adhering to the schema they were entered into the virtual observatory as, typically FGDC currently. In the future there may be more general, and more specific keys, such as 1-1 mapping between data and metadata parameter names. If there are formats, or other types of information you feel would be useful from the metadata service, please contact us. We cannot improve in this area without input from the user community. Send email to spidr-support@rt.ngdc.noaa.gov with your requests.

With all that said, constructing metadata requests is equally straightforward, and here are some examples.

Metadata Examples

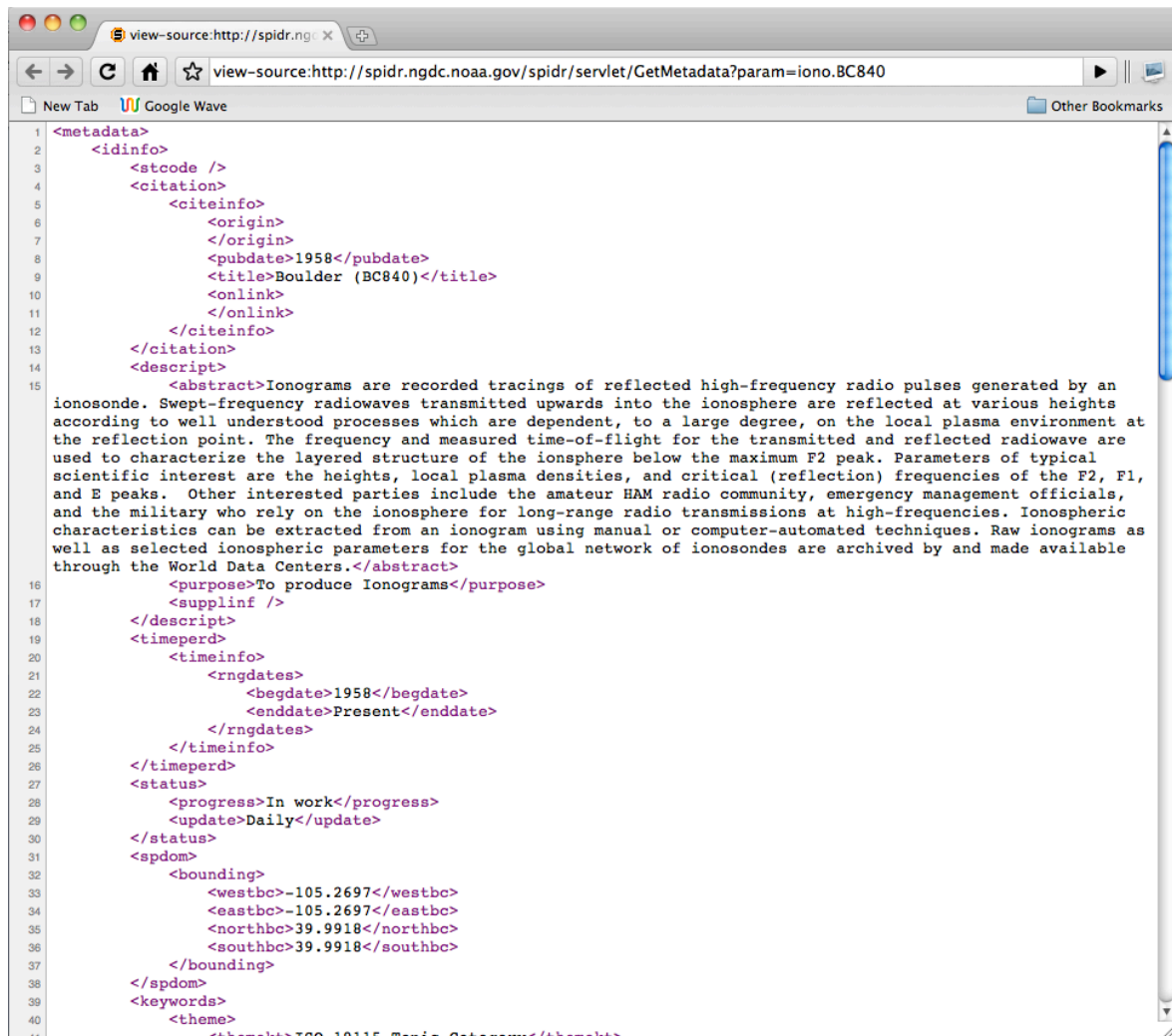
Here's the metadata associated with the Boulder ionosonde station, so the <query_string> is:

- param=iono.BC840

Complete URL/call

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetMetadata?param=iono.BC840>

Gives back an XML document containing all of the information pertinent to BC840 that looks like this in a web browser (obviously you'll want to parse it as XML in a program).

A screenshot of a web browser window displaying the source of an XML document. The browser's address bar shows the URL: http://spidr.ngdc.noaa.gov/spidr/servlet/GetMetadata?param=iono.BC840. The page content is an XML document with the following structure:

```
<metadata>
  <idinfo>
    <stcode />
    <citation>
      <citeinfo>
        <origin>
        </origin>
        <pubdate>1958</pubdate>
        <title>Boulder (BC840)</title>
        <onlink>
        </onlink>
      </citeinfo>
    </citation>
    <descript>
      <abstract>Ionograms are recorded tracings of reflected high-frequency radio pulses generated by an ionosonde. Swept-frequency radiowaves transmitted upwards into the ionosphere are reflected at various heights according to well understood processes which are dependent, to a large degree, on the local plasma environment at the reflection point. The frequency and measured time-of-flight for the transmitted and reflected radiowave are used to characterize the layered structure of the ionosphere below the maximum F2 peak. Parameters of typical scientific interest are the heights, local plasma densities, and critical (reflection) frequencies of the F2, F1, and E peaks. Other interested parties include the amateur HAM radio community, emergency management officials, and the military who rely on the ionosphere for long-range radio transmissions at high-frequencies. Ionospheric characteristics can be extracted from an ionogram using manual or computer-automated techniques. Raw ionograms as well as selected ionospheric parameters for the global network of ionosondes are archived by and made available through the World Data Centers.</abstract>
      <purpose>To produce Ionograms</purpose>
      <supplinf />
    </descript>
    <timeperd>
      <timeinfo>
        <rngdates>
          <begdate>1958</begdate>
          <enddate>Present</enddate>
        </rngdates>
      </timeinfo>
    </timeperd>
    <status>
      <progress>In work</progress>
      <update>Daily</update>
    </status>
    <spdom>
      <bounding>
        <westbc>-105.2697</westbc>
        <eastbc>-105.2697</eastbc>
        <northbc>39.9918</northbc>
        <southbc>39.9918</southbc>
      </bounding>
    </spdom>
    <keywords>
      <themekt>ISO_19115_Topic_Category</themekt>
    </keywords>
  </idinfo>
</metadata>
```

Example Basic REST Clients

There are currently three example clients, written in IDL, MATLAB, and Perl. They may serve as a starting point for subsequent client development, or you can simply use them as another means for obtaining data from SPIDR by incorporating and using them as libraries in another application. Each one provides another abstraction layer specific to the language for obtaining data from SPIDR.

Using these clients as a starting point, or as the basis for further development is very straightforward. Once you've obtained the code, incorporating it, and consequently SPIDR's data into your own applications is fairly easy.

IDL Rest Client

Using the IDL client, all it takes to obtain SPIDR data is the following one line of IDL code:

```
IDL> spidr = spidr_get_data( 'xs.goes11', [2008,1,1,0,0,0], [2008,1,31,23,59,59], /UNIX_TIME, verbosity=5 )
```

Thereafter, you can use 'spidr' as you would any other structure in your IDL. This client is independently maintained, as is its documentation.

[You can find more information regarding the IDL client here.](#)

MATLAB REST Client

The MATLAB/Octave client is similarly straightforward, and is available here: <https://sourceforge.net/projects/spidr-matlab/>

Python REST Client

This client is at the very beginning stages of development, but provides essential data and metadata features, it's available here:

<http://spidr.ngdc.noaa.gov/spidr/friend.do?hlink=http://code.google.com/p/spidr-python/>

Perl REST Client

This client is the least featureful of the trio, but shows how easy it is to create simple clients for consuming time series data.

It's only dependency is the LWP::Simple library, which is included with most Linux Perl distributions. Getting data is as simple as defining a function as follows

```
#####
#
# an example function that minimally wraps the REST GetData call with only
# required parameters. see the user's guide for the full complement.
#
#####
sub get_data ($$$$) {

    # assign args
    my ($param) = shift(@_);
    my ($dateFrom) = shift(@_);
    my ($dateTo) = shift(@_);
    my ($format) = shift(@_);

    # build the url
    my $url = $_DATA_PREFIX . "param=" . $param . "&format=" . $format .
        "&dateFrom=" . $dateFrom . "&dateTo=" . $dateTo;

    # execute, take appropriate action based on $format
    my $content = get($url);

    if ($format !~ /zip/) {
        print $content . "\n\n";
    } else {
        my $file = "${param}_${dateFrom}_${dateTo}.zip";
        open (FILE, ">$file") or die "Could not open file $file";
        print FILE $content;
        close(FILE);
        print "DONE. your content was stored in your current working directory, in file $file\n\n";
    }
}
}
```

[The entire client can be downloaded from SPIDR here](#)

Advanced Usage

In addition to the basic topics already covered, which focus on simple time series data access, SPIDR also provides more advanced capabilities and formats from its RESTful web services, the details of which follow. This advanced portion of the API provides different access mechanisms as well as formats, and adheres to different data standards as well, including the Common Data Model.

Common Data Model

The Common Data Model (CDM) is used to store multidimensional array data such as point observations (e.g. earthquakes), station time series (e.g. geomagnetic variations), trajectories (e.g. satellite tracks) and gridded fields (e.g. elevation model or air pressure variations with time on a latitude-longitude grid). CDM is derived from and is compatible with the NetCDF and HDF5 data models. The CDM object relations are shown in the Figure below.

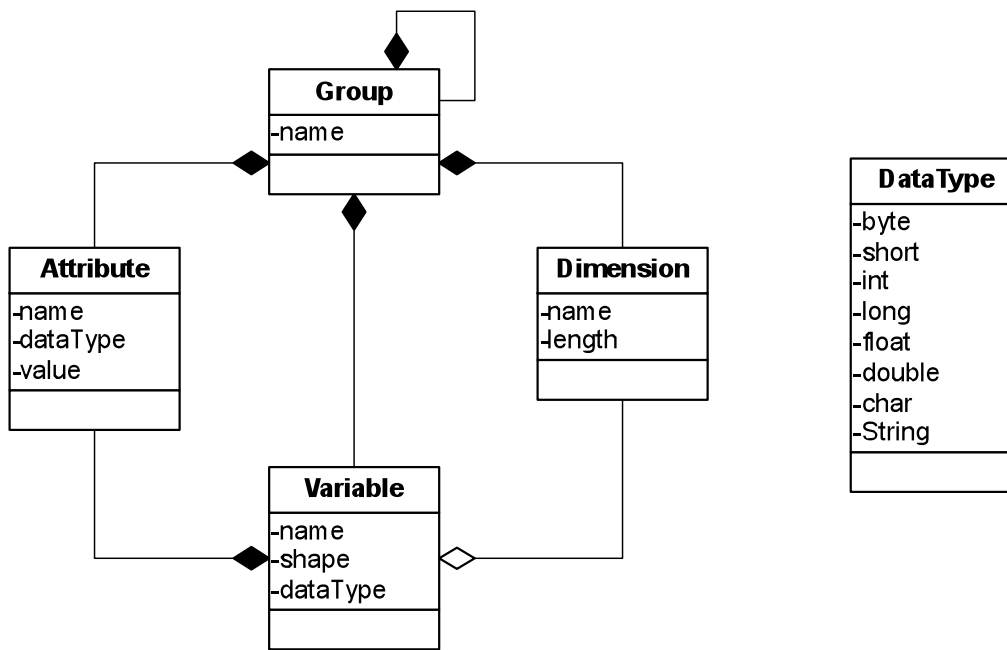


Figure 1. Object relations in Common Data Model (CDM)

The CDM objects definitions:

- *Group* is a container for other objects. It contains variables, dimensions, attributes, and other groups.
- *Variable* represents a multidimensional data array of specified type indexed by 1 or more dimensions. A variable may have 0 or more attributes, which contain additional metadata.
- *Dimension* is a named index used to describe the shape of the data array stored in a variable.
- *Attribute* is a name-value pair used to store additional metadata for groups and variables.

ESSE engine and data web services communicate data in CDM data objects in different formats. Inside the engine it is serializable Java object. For distributed data exchange binary NetCDF or its XML version NcML are used.

Common Query Language

To simplify for end user to query environmental data sources with different data models, such as CDM array, binary data granule, plot image or web map, we have created the Common Query Language (CQL). Instead of using different parameter set for each data source, as it was in the previous versions of the ESSE engine, the data query now is formed from a limited number of options, which are relevant to all the datasets integrated by ESSE. The QCL options can be encapsulated in the Java bean `DataRequestBean`, which is be passed to the DSI methods as an input parameter.

Below we illustrate the CQL data request options by examples of the REST data service calls. RESTful web services are more simple and less platform dependable compared to SOAP protocol, so everybody can access the REST service from a software client or by REST request URL in a web browser.

The current version of CQL supports the following data query options:

Parameter	Description	Values
command	command to be sent to the data service	<ul style="list-style-type: none"> get (default) – queries the getData activity; provides the main functionality for getting data from service describe – queries the resource metadata in XML format and shows an HTML page
dataset	specifies parameter name to query in a resource specific format	param[.vlevel[. #]]@theme[.source] geom_z@Geom geom_y.Geom_yr@Geom.yr SkinTemperature.Surface@Weather F13200512311748@DMSP.OIS
datefrom	date-time before or equal to the first data sample	yyyy-mm-ddThh:mm:ssUTC
dateto	date-time after or equal to the last data sample	yyyy-mm-ddThh:mm:ssUTC
location	sets spatial constraints for data query	<ul style="list-style-type: none"> point: (39.0,27.5) station: BOU
format	format for data export	xml, ncml, echo(dmosp gralules list), jpg, png, tif, ...
async	sets asynchronous mode for request	true enables the async requests
asyncorder	ID of the asynchronous data order to check its status or get result	

Asynchronous RESTful Data Service

Selection of large chunks of data from a database or a tape library may require too much memory or can take so long that the web service call will time out. The error can be either “not enough memory” runtime exception or a web server timeout. To avoid the memory size error in a synchronous web service we have introduced a throttling function which will estimate the size of the data chunk before the actual data selection, and report an error to the client if the data size is too large. However, this throttling mechanism is limiting the use of the RESTful services, especially for tape library archive, where data selection takes several minutes to execute.

One solution for the time delay and the data size limitations is an asynchronous RESTful data service. In this case we can split the large selection data into smaller parts which will fit into server memory, and a client will receive the selection by calling the server several times, each time receiving only a part of the overall request. The asynchronous data service protocol becomes stateful and requires several types of messages, including request ID, request status, and data. At the first call, the data service will return to client the data request ID. Using that ID, a client application can repeatedly call the service again and again, each time receiving from the server either a status report for the request, or a small chunk of the requested data.

Architecture of the asynchronous RESTful data service has three tiers: REST service container, OGSA-DAI data resource container, and a persistence storage backend. The REST service is implemented as a Java servlet. Its purpose is maintain the asynchronous session with the client, to translate HTTP get requests from client into SOAP requests to the OGSA-DAI data resource, and to pass the OGSA-DAI service reply back to the client. The OGSA-DAI data resource provides special activities for data querying and processing. It also has a queue to store state of the data request, and to data pools. A disk data pool is used to work with the “slow” tape library. “In memory” data pool is used to store small chunks of data received from the database backend. The storage backend can be either a database or a tape library.

At the first call, the asynchronous RESTful data service always returns to client the data request ID. Behind the scenes the service opens a stateful session for the client and redirects the request to the OGSA-DAI activity, which will put the request into a queue. The client may specify that the returned data should be split into smaller chunks. Then the OGSA-DAI activity will put multiple requests into the queue, each for a separate chunk. A queue monitor inside the OGSA-DAI container will check for free space in the data pool and for data selection requests in the queue. If the pool is not full and the queue is not empty, it will select a chunk of data from storage, save it in the pool, and delete the request from the queue.

Each subsequent client call in the asynchronous session must refer to the same request ID. The RESTful service reply will depend on the session status. If the requested data is not ready yet, the service will return status “processing”. If the data is not available, the service will return status “error” and close the session. If the session was closed or does not exist, the service returns “order <ID> not found”.

When the data selection gets ready, the next client call will receive from the server a web link to the data granule in the disk pool or a chunk of data from memory pool. When all the selected data will be sent to client, the asynchronous data request session will be closed. After that the service call with the same request ID will return status “EOF”.

An example of the asynchronous data request to tape library storage is shown in figure 2. This is a typical use case for remote sensing applications, where the satellite images from different orbits are stored as binary granules on a robotic tape library. Image transfer from the library to the online disk pool takes tens of seconds, so the asynchronous service is used to avoid timeouts. The first call to service returns request ID and initiates data transfer from tape to disk. The next service calls for this request ID will return status “processing” until the image will be copied from tape to disk. After that a service call will return a link to the image on disk and close the session.

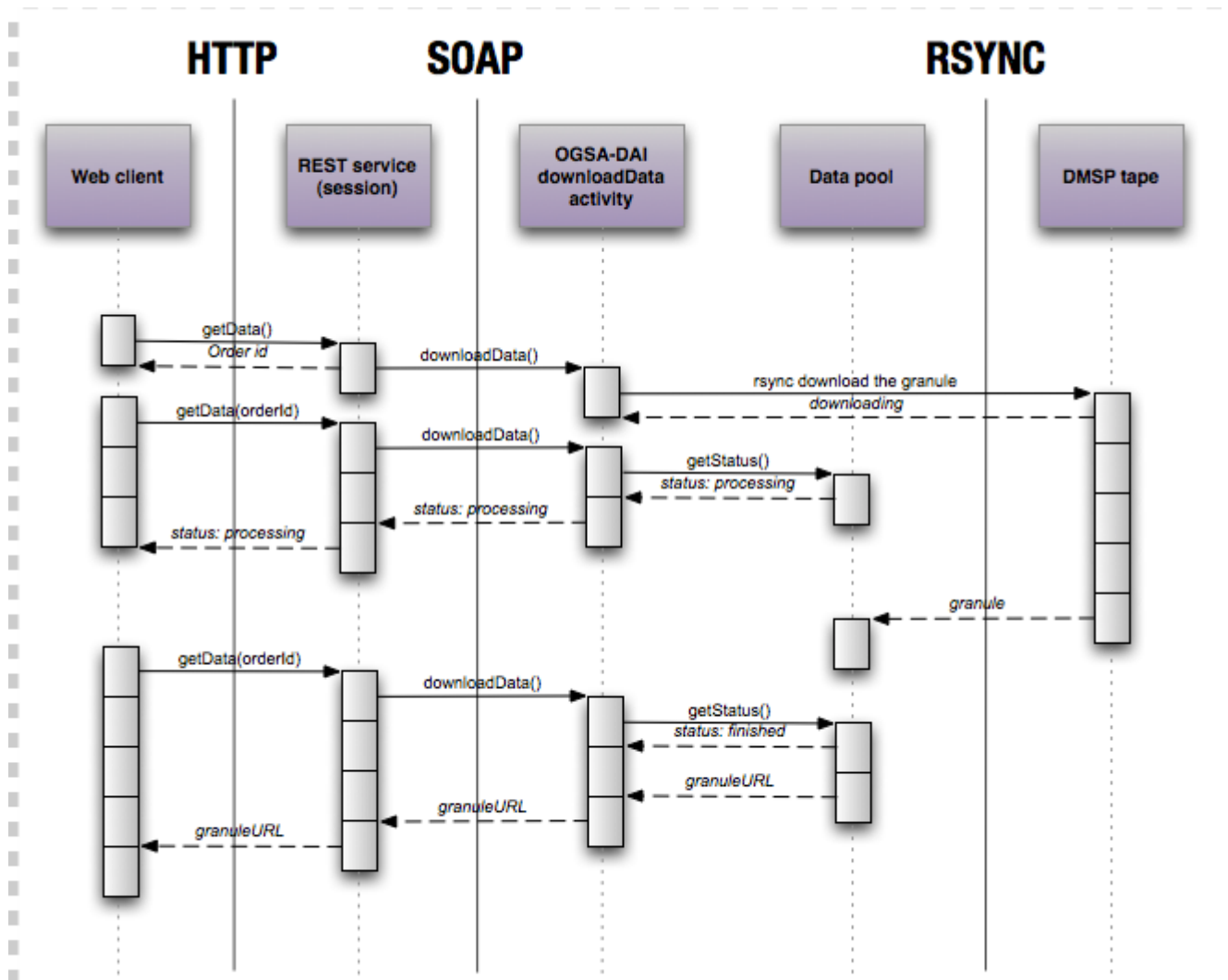


Figure 2. Asynchronous data request to tape library

Streaming data service request can be used to select very large subset from a database. For example, it can be time series for a long date interval from many stations. In this case the OGSA-DAI activity `getStreamData` will split the single request from a client into multiple requests to the database, each for one station and a streaming time window (a day, a month, etc.). When called from the client, the RESTful data service will return a chunk of the whole subset. The client should call the service as many times as the number of chunks in the subset (= number of stations X number of time subintervals). For a gridded dataset, the total number of chunks will be the number of grid points in the Region of Interest multiplied by the number of time subintervals in the request time range (figure 3).

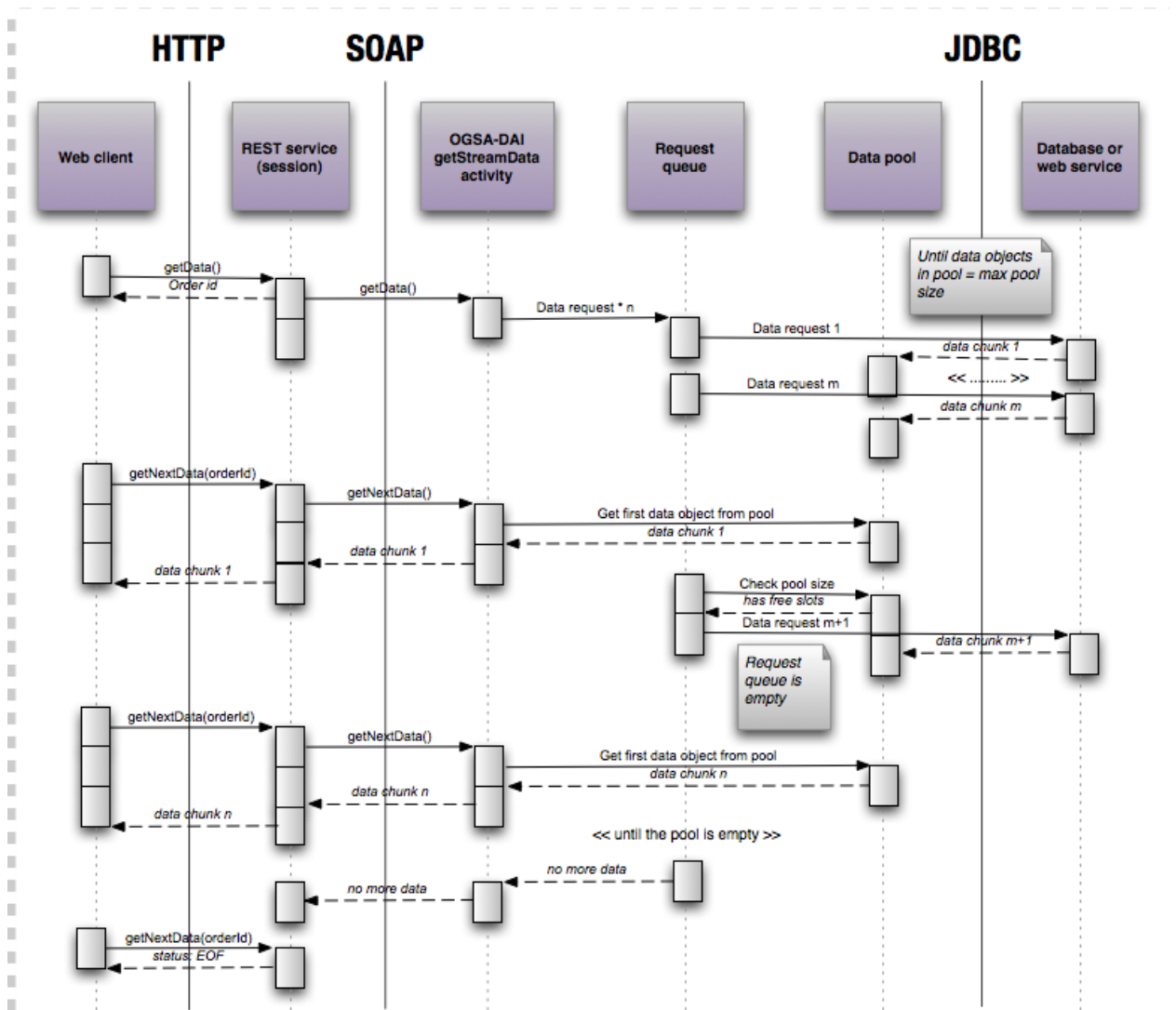


Figure 3. Asynchronous streaming RESTful data request to a database

SPIDR Advanced RESTful Web Service By Example

Metadata

Get the list of available data sets:

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2>

Get the metadata for a given data set:

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?command=describe&dataset=@Geom>

Synchronous get data

Get SPIDR time series

A. For minute geomagnetic variations

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?format=xml&datefrom=2001-01-01T00:00:00UTC&dateto=2001-01-05T00:00:00UTC&dataset=geom_y@Geom&location=BOU

B. For hourly geomagnetic variations

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?format=xml&datefrom=2001-05-01T00:00:00UTC&dateto=2001-05-03T00:00:00UTC&dataset=geom_z@Geom.hr&location=AIA

C. For IMF By

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?dataset=imf_by@ImfMin&location=ace&format=xml&datefrom=2001-01-01T00:00:00UTC&dateto=2001-01-03T00:00:00UTC

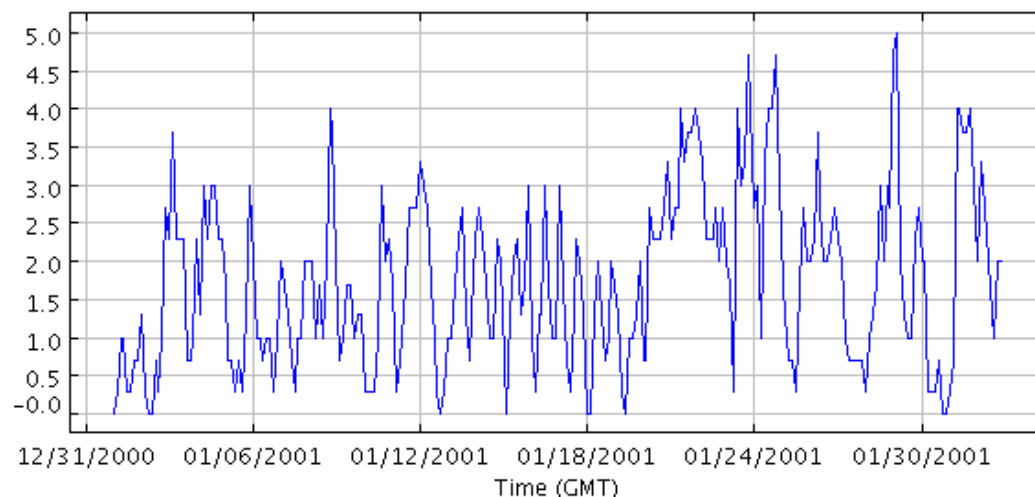
D. For Kp index

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?dataset=index_kp@KpAp&format=xml&location=ALL&datefrom=2001-01-01T00:00:00UTC&dateto=2001-02-01T23:59:00UTC

Get a SPIDR time series plot

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?dataset=index_kp@KpAp&format=jpg&location=ALL&datefrom=2001-01-01T00:00:00UTC&dateto=2001-02-01T23:59:00UTC

Which generates:



Get DART time series data

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?format=xml&datefrom=2000-05-01T00:00:00UTC&dateto=2000-05-03T00:00:00UTC&dataset=Temperature.Surface@Dart&location=D157_1999

Get time series data from NWS weather forecast

[http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?format=xml&datefrom=2010-03-01T00:00:00UTC&dateto=2010-03-03T00:00:00UTC&dataset=SkinTemperature.Surface@Weather&location=\(56,38\)](http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?format=xml&datefrom=2010-03-01T00:00:00UTC&dateto=2010-03-03T00:00:00UTC&dataset=SkinTemperature.Surface@Weather&location=(56,38))

Get DMSP granules list

[http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?datefrom=2006-01-07T00:00:00UTC&dateto=2006-01-14T23:59:59UTC&dataset=F13@DMSP&location=\(55.0,42.0\)&format=echo](http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?datefrom=2006-01-07T00:00:00UTC&dateto=2006-01-14T23:59:59UTC&dataset=F13@DMSP&location=(55.0,42.0)&format=echo)

Order DMSP granule for download (synchronously)

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?dataset=F13200512311748@DMSP.J4>

Asynchronous Data Requests

Order DMSP granule for download (asynchronously):

<http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?dataset=F13200512311748@DMSP.J4&asyncc=true>

Check DMSP order status & get the result when finished:

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?dataset=F13200512311748@DMSP.J4&asyncc=true&asynccorder=ORDER_ID

One minute geomagnetic variations, streaming by 1 day, asynchronously:

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?format=ncml&datefrom=2001-01-01T00:00:00UTC&dateto=2001-01-05T00:00:00UTC&dataset=geom_z@Geom&location=BOU×hift=d1&async=true

Get the data for the previous asynchronous request, returns a daily data chunk for each service call

http://spidr.ngdc.noaa.gov/spidr/servlet/GetData2?format=ncml&dataset=geom_z@Geom&asyncc=true&asynccorder=ORDER_ID